Зберігання, аналіз та захист даних в інформаційних системах

УДК 003.26+004.051

М.И. Верещак, А.В. Неласая

Запорожский национальный технический университет, Запорожье

ОСОБЕННОСТИ РЕАЛИЗАЦИИ БИБЛИОТЕКИ АРИФМЕТИКИ ПРОИЗВОЛЬНОЙ ТОЧНОСТИ НА ГРАФИЧЕСКИХ УСКОРИТЕЛЯХ ДЛЯ КРИПТОГРАФИЧЕСКИХ ПРИЛОЖЕНИЙ

В работе рассматриваются ключевые моменты реализации библиотеки арифметики произвольной точности под технологию CUDA для использования в криптографических приложениях. Предлагаются оригинальные подходы к решению таких проблем, как ограничение быстродействия скоростью доступа к памяти, сложность параллельной реализации алгоритмов простых арифметических операций, необходимость синхронизации между блоками параллельных нитей. Проведены тесты сравнения эффективности простых арифметических операций (векторного сложения и нахождения суммы арифметической прогрессии) на CPU и GPU. Рассмотрены нюансы реализации операции сложения целых длинных чисел на GPU.

Ключевые слова: *арифметика произвольной точности, GPU, CUDA, параллельные алгоритмы, векторный вычислитель, графический ускоритель.*

Введение

Одним из наиболее важных элементов современных криптографических систем, а также краеугольным камнем исследований в области криптографии являются библиотеки длинных чисел, на которых основана реализация алгоритмов криптографии. Как было выявлено в результате исследований [1], использование библиотеки длинной арифметики снижает быстродействие алгоритмов в 6-10 раз вследствие большого числа обращений к оперативной памяти, и увеличивает объем расходуемой оперативной памяти в 4-5 раз, по сравнению с реализацией без использования этих библиотек. В результате таких накладных расходов на обеспечение работы длинной арифметики математическая оценка сложности некоторых алгоритмов не совпадает с получаемыми на практике результатами их работы.

Основное конечное поле, над которым определяются базовые криптографические объекты, например эллиптические кривые, зачастую имеет размер модуля порядка 160 – 1024 бита. Такие значения длин параметров в совокупности с тщательным отбором стойких кривых обеспечивают достаточную криптографическую стойкость базовых криптографических примитивов. Основные задачи эллиптической криптографии – это реализация процедур генерации ключей, формирования и проверки цифровой подписи, а также выбора параметров криптографической системы. Последняя задача включает, в частности, задачи определения порядка кривой и выбор кривой, стойкой к известным методам криптоанализа. Они являются достаточно сложными и ресурсоемкими.

Традиционно, такие задачи удобно решать с использованием параллельных методов на мощных вычислительных кластерах, суперкомпьютерах, использовать технологию GRID. Однако, в последнее время предпочтение отдается решениям, которые претендуют на название «персональный суперкомпьютер». Их основное отличие состоит в том, что персональные суперкомпьютеры помещаются в корпус десктопа или стоечного сервера и могут быть установлены непосредственно на рабочем столе, а не в специально оборудованных помещениях, как того требуют кластерные суперкомпьютеры [2].

В последние несколько лет перспективным направлением увеличения вычислительной мощности компьютерных систем стал перенос вычислений на графические процессоры (GPU – от англ. graphics processing units). Эту концепцию, получившую аббревиатуру GPGPU (от англ. general purpose graphics processing units), поддержали основные производители графических ускорителей, в частности компания NVIDIA, разработавшая технологию CUDA и соответствующую спецификацию [3], облегчающую процесс создания алгоритмов для GPU.

Корпорация NVIDIA придала новое значение понятию «персональний суперкомпьютер» благодаря технологии высокопродуктивных вычислений на графических ускорителях GPU Tesla. Технология NVIDIA CUDA™ – это единственная среда разработки на C, которая позволяет программистам и разработчикам писать программное обеспечение, на порядки ускоряющее сложные вычислительные

задачи благодаря многоядерной вычислительной мощности графических процессоров.

На данный момент в городе Дубна открыт научно-образовательный центр "Параллельные Вычисления" [4], который при решении многих, как научных, так и бизнес - задач, где требуется создание не просто программного обеспечения, а системы с сильной математической составляющей, способной производить сложные расчеты и совершать действия, исходя из полученных результатов, оказывает услуги по анализу ситуации, разработке математической модели, построению соответствующего вычислительного алгоритма, его тестированию и адаптации к архитектуре будущей системы, дальнейшей разработке программы для использования алгоритма. Наиболее значимые проекты центра: CBDA: Cyclotron Beam Dynamics Analysis; задача "Клеточного автомата"; уравнение "Теплопроводности"; задача обработки сигнала в режиме реального времени.

Однако вышеперечисленные задачи используют вычисления с вещественными числами, тогда как криптографические алгоритмы оперируют с математическими объектами, определенными над простыми и расширенными конечными полями. Базовым уровнем для таких вычислений являются модульные арифметические операции с длинными целыми числами, а также полиномиальная арифметика, в частности арифметика расширенного поля Галуа характеристики два. Решенной задачи в такой постановке авторы данной статьи не нашли.

Постановка задачи. Популярность и интенсивное развитие технологии CUDA, а также отсутствие готовых решений для применения в криптографических целях привели к идее создания библиотеки длинной модульной целочисленной и полиномиальной арифметики, использующей GPU в качестве основного вычислительного ядра. Главная особенность и преимущество GPU перед CPU это высокая степень параллелизма вычислений - современные GPU содержат от нескольких десятков, до нескольких сотен скалярных процессоров, которые работают в параллельном режиме. Также GPU имеет свою внутреннюю память, которая обладает более высоким быстродействием по сравнению с RAM.

Преимущества графических процессоров оборачиваются основной проблемой при составлении алгоритмов, так как для того, чтобы достичь высокой эффективности при таких массивно-параллельных вычислениях, нужно учитывать множество факторов: архитектурные особенности GPU, быстродействие и порядок доступа к памяти, механизмы синхронизации между вычислительными потоками. Важную роль играет также приспособленность самого алгоритма к исполнению в параллельном режиме.

Можно выделить такие основные ключевые моменты создания библиотеки длинной арифметики на GPU:

– скорость доступа к памяти;

– параллельная реализация алгоритмов простых арифметических операций;

– синхронизация между блоками параллельных нитей.

Далее рассмотрим более подробно проблемы, возникающие при работе с GPU.

Основной материал

Работа с памятью. Выполняемый на GPU код не имеет доступа к основной оперативной памяти компьютера, а код, выполняемый на CPU, не имеет доступ к внутренней памяти GPU. Эта особенность архитектуры GPU добавляет дополнительное звено в процесс вычислений – копирование исходных данных во внутреннюю память GPU, и копирование результата вычислений в оперативную память. Скорость выполнения этой операции ограничена быстродействием RAM, что приводит к значительному увеличению накладных расходов. Очевидно также, что имеют место временные затраты на инициализацию GPU.

Параллельные алгоритмы. Основной подход в реализации параллельных алгоритмов арифметических операций заключается в использовании векторных вычислительных устройств. Длинное целое число может быть представлено как вектор, элементами которого являются разряды этого числа. Под разрядами подразумевается часть числа, размер которой соответствует машинному слову GPU. В таком случае множество процессоров GPU можно рассматривать как векторное вычислительное устройство. Каждый процессор при этом будет обрабатывать по одному разряду каждого операнда. При такой реализации наиболее весомой является проблема переноса при переполнении разряда. В случае с GPU эта проблема усугубляется тем, что отдельные потоки на GPU обладают ограниченным возможностями для обмена данными друг с другом.

Синхронизация между потоками. Для двух потоков, выполняющихся на GPU, не всегда возможно заранее определить будут ли они реально выполняться параллельно. Эта особенность архитектуры графических процессоров накладывает ограничения на возможности синхронизации между потоками, обмена данными и контроля очередности исполнения кода. Потоки, выполняемые на GPU, называются нитями. Нити объединены в блоки. Какие именно из нитей блока выполняются одновременно предсказать нельзя. Единицей параллельно выполняемых нитей является warp. Размер warp'a зависит от архитектуры конкретной модели графического ускорителя. Одна задача может быть распределена на несколько блоков [5].

Тесты производительности. Чтобы оценить степень влияния операций доступа к памяти на общее время выполнения задачи, авторами было разработано соответствующее программное обеспечение и проведены тесты, суть которых заключается в сравнении времени выполнения простых операций над большим массивом данных на CPU и на GPU.

Первый тест – это операция векторного сложения. В качестве исходных данных выступают два одномерных равных по величине массива чисел с плавающей точкой. Результат работы программы – также одномерный массив, содержащих суммы элементов первых двух массивов. При этом каждая операция сложения на GPU выполняется отдельной нитью, а на CPU – в цикле в одном потоке. Полученные авторами значения времени выполнения приведены в табл. 1.

Второй тест – это построение таблицы значений сумм элементов арифметической прогрессии. В результате работы программы создается одномерный массив вещественных чисел, n-й элемент которого является суммой n первых чисел арифметической прогрессии. Каждая нить на GPU создает свой элемент массива, вычисляя полную сумму прогрессии. Для получения достоверных данных сравнения при реализации этого же алгоритма на CPU не проводилось никаких оптимизаций. На каждой итерации вычислялась полная сумма прогрессии без учета значений, полученных на предыдущей итерации. В табл. 2 приводятся результаты тестирования.

В обоих тестах процесс вычисления на GPU повторялся дважды с целью выявить существование дополнительных расходов ресурсов на инициализацию графического ускорителя. Параметры компьютера, на котором запускались тесты: процессор: AMD Phenom II X4 840 (3640 MHz), память RAM: 2 Gb, видеокарта Nvidia GeForce 210.

Таблица 1

Результаты теста времени выполнения операция векторного сложения

Объем данных (кол. эл.)	Время выполнения		
	CPU (мс)	Прогон 1. GPU(мс)	Прогон 2. GPU (мс)
1024	0	294	0
8192	0	302	0
65536	0	291	0
131072	1	288	2
262144	2	305	2
1048576	5	300	8

Таблица 2

Результаты теста времени выполнения вычисления сумм элементов арифметической прогрессии

Объем данных (кол. эл.)	Время выполнения		
	CPU (мс)	Прогон 1. GPU(мс)	Прогон 2. GPU (мс)
1024	0	298	0
2048	2	295	1
4096	9	298	5
8192	37	321	21
16384	148	377	86
32768	590	632	341
65536	2361	1663	1365
131072	9443	5750	5455
262144	37769	22105	21815

Полученные результаты подтверждают высказанные предположения: по мере увеличения количества операций, исполняемых на GPU, относительно объема обрабатываемых данных, соотношение скорости выполнения алгоритмов на CPU и GPU изменяется в пользу последнего. Также подтвердилось существование задержки, требующейся для инициализации GPU, которая является постоянной величиной для данного оборудования и не зависит от объема исходных данных. В проведенных тестах она составила 295 – 305 мс.

Таким образом, основные пути повышения эффективности библиотеки длинной арифметики для GPU - это уменьшение количества операций обмена

данными между основной оперативной памятью и внутренней памятью GPU, а также максимальное использование параллелизма.

Оптимизировать работу с памятью можно двумя способами.

Во-первых, внутреннее представление длинных чисел должно быть двойным. Одна копия числа содержится в оперативной памяти компьютера, а вторая – во внутренней памяти GPU. При этом обе копии никогда не создаются одновременно. Исходные данные размещаются в оперативной памяти, и их копирование во внутреннюю память GPU происходит лишь в момент передачи числа в качестве операнда для выполнения вычислений. Аналогичным образом результаты вычислений остаются во внутренней памяти GPU до тех пор, пока не возникнет необходимость вывести этот результат на печать. Также в отношении чисел этой библиотеки можно применить подход, аналогичный используемому в языке программирования LISP(для всех данных), или в JAVA (для работы со строковыми данными). Суть подхода заключается в том, что объект, в данном случае число, содержит адрес размещения данных, представляющих собой немодифицируемое значение этого числа. В таком случае отпадает необходимость в обеспечении механизма синхронизации данных между RAM и памятью GPU в случае их изменения.

Во-вторых, для оптимизации работы с памятью имеет смысл отказаться от классического подхода описания математических выражений в пользу постфиксной (или так называемой "обратной польской") нотации. На сегодняшний день такой синтаксис применяется в консольном калькуляторе unix-подобных систем dc, а также в языке программирования FORT. Последний интересен также с точки зрения подхода к организации памяти. В FORT применяется стековая структура хранения данных. При вводе нового операнда он помещается в вершину стека. Любые операции выполняются над операндами, находящимися в этом стеке, сюда же сохраняется результат выполнения операций. Применительно к GPU, если организовать стек во внутренней памяти, можно получить дополнительный выигрыш за счет уменьшения количества операций копирования операндов.

В случае с параллельной организацией алгоритмов вычислений какой-либо общий принцип отсутствует, и особенности реализации зависят от конкретной задачи. В данной статье будут рассмотрены параллельные алгоритмы сложения и умножения.

Алгоритм сложения. Как было сказано, основная проблема при сложении длинных чисел – сохранение флага переноса между разрядами. Рассмотрим два возможных варианта реализации параллельного алгоритма сложения.

Первый вариант заключается в поразрядном сложении чисел, где под разрядом понимается наименьшая единица внутреннего представления числа (2, 4 или 8 байт в зависимости от архитектуры). При таком подходе каждый отдельный поток на GPU выполняет лишь одну операцию сложения двух разрядов операндов. Для сохранения флага переноса можно применить глобальную память GPU (доступную всем нитям), разделяемую память GPU (доступную нитям одного блока), либо неполное использование разряда во внутреннем представлении числа на GPU (при 32-битном внутреннем представлении только младшие 24 бита хранят значение числа, а старшие 8 используются для сохранения переноса при переполнении разряда).

Глобальная память удобна с точки зрения простоты обращения и имеет достаточно большой объем, сопоставимый с объемом оперативной памяти компьютера. Недостатком ее является высокая латентность доступа. Несмотря на то, что скорость обращения к ней на порядок превышает скорость обмена с оперативной памятью и CPU, время доступа к ней значительно ниже, чем при работе с регистровой памятью мультипроцессора, которая используется для хранения данных в разделяемой памяти.

Разделяемая память является наилучшим решением с точки зрения удобства использования и скорости обращения, однако она является ограниченным ресурсом (не более 16 Кбайт на блок) и доступна только для нитей одного блока. Следовательно, при ее использовании возникает два ограничения: одна математическая операция должна выполняться только лишь одним блоком нитей, и максимальная длина операндов ограничивается объемом доступной разделяемой памяти.

При неполном использовании разрядов имеем два основных недостатка: увеличение объема памяти для внутреннего представления числа (из доступных 32 разрядов только часть является значащими) и более сложные механизмы учета переноса, требующие выполнения дополнительных битовых операций над операндами.

Второй вариант реализации алгоритма сложения заключается в том, что каждая нить использует не один, а несколько разрядов операнда [6]. При таком подходе вместо хранения флагов переноса результатов сложения каждой пары разрядов, достаточно хранить лишь один флаг переноса для каждого блока разрядов. В таком случае объема разделяемой памяти достаточно для хранения всех флагов переноса, и необходимость в применении описанных выше подходов для решения этой задачи отпадает. Также можно ограничить запуск вычислительного ядра одним блоком нитей, что предоставляет дополнительные возможности синхронизации между нитями.

С учетом всех преимуществ и недостатков представленных путей реализации, для алгоритма

сложения был выбран второй способ с хранением флага переноса в разделяемой памяти GPU.

Алгоритм умножения. Для умножения длинных чисел уже разработано несколько оригинальных параллельных алгоритмов. В статье [7] произведен анализ быстродействия этих алгоритмов. Исходя из результатов, полученных в [7], наибольшей производительностью обладают параллельные реализации алгоритмов Карацубы и "быстрый столбик".

Принимая во внимание то, что алгоритм Карацубы дает преимущество лишь для чисел достаточно большой длины (более 2048 бит), и при этом, исходя из результатов сравнения, "быстрый столбик" лишь незначительно уступает ему в скорости, для реализации операции умножения в библиотеке для CUDA был выбран алгоритм умножения "быстрый столбик".

При реализации операции умножения на графических процессорах существует множество нюансов, некоторые из которых были рассмотрены при оценке алгоритмов сложения.

Полный анализ данной задачи не соответствует формату данной статьи и будет рассмотрен отдельно.

Выводы

Реализация библиотеки длинной арифметики на GPU достаточно перспективное направление. Перенос простых арифметических операций на GPU дает выигрыш в производительности по сравнению с CPU при использовании даже самых простых моделей графических адаптеров. На этом пути существует еще множество возможностей для исследований архитектурных особенностей графических процессоров и оптимизаций с их учетом.

Дальнейшие исследования будут направлены на практическую реализацию арифметики длинных чисел, в частности модульной и полиномиальной арифметики с применением как классической, так и постфиксной нотации в интерфейсах построения арифметических выражений.

Список литературы

1. Неласая А.В. Оценка эффективности вычислений в библиотеках длинной арифметики / А.В. Неласая, М.И. Верещак // Проблеми і перспективи розвитку IT-індустрії: II Міжнародна науково-практична конференція, 18 – 19 листопада 2010 р. – Харків. – С. 226-227.

2. Конференция "HPC DAY 2010: Сучасні рішення для високопродуктивних обчислень" [Электронный ресурс]. – К.. – Режим доступа к ресурсу: http://www.ustar.ua.

3. Что такое CUDA? [Электронный ресурс]. – Режим доступа к ресурсу:http://www.nvidia.ru/object/what_is_cuda_new_ru.html

4. Научно-образовательный центр "Параллельные вычисления" [Электронный ресурс]. – Режим доступа к ресурсу: http://www.parallel-compute.com.

5. Борсеков А.В. Основы работы с технологией CUDA / А.В. Борсеков, А.А. Харламов . – М.: ДМК Пресс, 2010. – 232 с.

6. Нестеров А.П. Параллельный алгоритм сложения чисел большой разрядности [Электронный ресурс] / А.П. Нестеров, А.С. Федулов. – Режим доступа к ресурсу: http://network-journal.mpei.ac.ru/cgi-bin/.

7. Качко Е.Г. Распараллеливание алгоритмов умножения чисел многократной точности [Электронный ресурс] / Е.Г. Качко // Параллельные вычислительные технологии 2011: Международная научная конференция; Московский государственный университет имени М.В. Ломоносова, г. Москва, 28 марта – 1 апреля 2011.. – Режим доступа к ресурсу: http://kvap.ru/pavt2011_cd/short/015.pdf

Поступила в редколлегию 19.09.2011

Рецензент: д-р техн. наук, проф. Л.М. Карпуков, Запорізький національний технічний університет, Запоріжжя.

ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ БІБЛІОТЕКИ АРИФМЕТИКИ ДОВІЛЬНОЇ ТОЧНОСТІ НА ГРАФІЧНИХ ПРИСКОРЮВАЧАХ ДЛЯ КРИПТОГРАФІЧНИХ ДОДАТКІВ

М.І. Верещак, Г.В. Неласа

У роботі розглядаються ключові моменти реалізації бібліотеки арифметики довільної точності під технологію CUDA для використання в криптографічних додатках. Пропонуються оригінальні підходи до вирішення таких проблем, як обмеження швидкодії швидкістю доступу до пам'яті, складність паралельної реалізації алгоритмів простих арифметичних операцій, необхідність синхронізації між блоками паралельних ниток. Проведено тести порівняння ефективності простих арифметичних операцій (векторного додавання і знаходження суми арифметичної прогресії) на CPU та GPU. Розглянуто нюанси реалізації операції додавання довгих цілих чисел на GPU.

Ключові слова: *арифметика довільної точності, GPU, CUDA, паралельні алгоритми, векторний обчислювач, графічний прискорювач.*

IMPLEMENTATION DETAILS OF THE ARBITRARY-PRECISION ARITHMETIC LIBRARY ON GRAPHICS ACCELERATORS FOR CRYPTOGRAPHIC APPLICATIONS

M.I. Vereschak, G.V. Nelasa

In this paper the midpoints of implementation of arbitrary-precision arithmetic library with using CUDA technology for cryptographic applications are considered. The ingenious approaches to solving problems such as limited speed of memory access, complexity of parallel implementation of algorithms for basic arithmetic operations and synchronization between the blocks of parallel threads are proposed. Tests to compare the effectiveness of simple arithmetic operations (vector addition and calculation the arithmetic progression sum) on the CPU and GPU were conducted. The nuances of implementation the adding operation for long integer numbers using GPU are examined.

Keywords: *arbitrary-precision arithmetic, GPU, CUDA, parallel algorithms, a vector computer, graphically accelerator.*